Ontology-based Reconfiguration Agent for Intelligent Mechatronic Systems in Flexible Manufacturing

Yazen Alsafi, Valeriy Vyatkin

University of Auckland, Dept. of Electrical and Computer Engineering,

y.alsafI@ieee.org, v.vyatkin@auckland.ac.nz

Abstract-This paper proposes an approach to achieving fast reconfiguration of modular manufacturing systems, based on an ontology-based reconfiguration agent. The agent uses ontological knowledge of the manufacturing environment for the purpose of reconfiguration without human intervention. The current mass customization era requires increased flexibility and agility in the manufacturing systems to adapt changes in manufacturing requirements and environments. Our configuration agent minimises the overheads of the current reconfiguration process by automating it. It infers facts about the manufacturing environment from the ontological knowledge model and then decides whether the current environment can support the given manufacturing requirements. This paper proposes the agent architecture enabling the integration between the high level planning with the distributed low level control compliant with the upcoming IEC 61499 function blocks standard.

I. Introduction

The rise of the global economy has generated a greater competition for the manufacturing enterprise from every part of the globe. Nowadays, to be competitive, manufacturers must not only produce low-priced products with high quality, but also manufacture products that need to be customised according to the customer’s personal preferences. Consequently, these requirements take the reasonability of increasing the product variants and decreasing the product life cycles. The consequences of these trends creates a need of building more efficient, flexible and agile manufacturing systems that can adapt new and changing requirements or changes in the manufacturing environment. These systems will allow manufacturers to produce multiple variations of customised products at the price of standardised mass products [1].

In this paper we propose an ontology-based reconfiguration agent that attempts to produce a new reconfiguration of the system reacting on changes in the requirements or the manufacturing environment. The agent forms a new reconfiguration after analysing the new requirements and inferring facts about the environment to deduce whether the current environment can handle the given requirements. The benefit of this approach is minimizing the overheads of the reconfiguration process by achieving rapid reconfiguration with minimum human intervention.

A methodology is proposed to integrate the high level planning with the low level control of the mechatronic system by using a deployment manager that manages the deployment process of the new reconfiguration on the embedded level of the system.

Rest of the paper is organised as follows: Section II defines the reconfiguration challenge in the manufacturing industry and introduces our proposed solution to this challenge while Section III introduces the notion of ontological knowledge representation and Section IV discusses the knowledge model of our simple manufacturing environment model. Section V introduces a reconfiguration agent and then discusses its architecture and implementation, while Section VI gives an insight on our future work. Finally in Section VII conclusions are drawn.

II. Reconfiguration Challenge

One of the main challenges faced by the manufacturing industries is rapid reconfiguration of manufacturing systems to handle rapid change in business environment without human intervention. One important criteria of the manufacturing factory is the flexibility characteristic of producing multiple variations of customised products. The factory must be flexible enough to support different sequences of production as well as allowing changes in the production system for new products offerings [1].

A number of works, addressing the reconfiguration problem, were departing from the fact, that the mechatronic components, or machines, have autonomous control and resemble robots in behaviour [2]. When placed together they are capable of advertising own services and discovering services of the other machines in the group. In this paper we are looking at the problem from a different angle: if the capabilities of individual machines are described in some formal form, how to infer the capabilities of their aggregation? Would it be possible to specify the requirements to a missing component, provided that some machines are already supplied and specification of the desired product is given?

In this paper we present a methodology based on the ontology-based reconfiguration agent [3]. This methodology can reduce some of the overhead mentioned above by rapidly reconfiguring the system via inferring useful facts from the ontological descriptions of the manufacturing environment. This approach is further extended by deployment manager that deploys the new configuration produced by the reconfiguration agent on the distributed controllers of the system.

The proposed methodology can reduce some of the overheads mentioned above by rapidly reconfiguring the system via utilising the ontological descriptions of the manufacturing environment and using reconfiguration agent that infer useful facts from these descriptions. Moreover the reconfiguration agent analyses the given requirement to check if the manufacturing environment can handle it.

III. RELATED WORKS

A reconfigurable manufacturing system (RMS) is defined as “a system designed at the outset for rapid change in structure, as well as in hardware and software components, in order to quickly adjust production capacity and functionality within a part family” [4].

The RMS concept of living and evolving factories, that quickly adapt new products and changing market demands, was introduced at Engineering Research Centre of the University of Michigan (UM) in the mid 1990s. Subsequently, RMS enabling technologies were developed at both the UM and in Europe and Canada. RMS is being recognised today as a necessary tool for increasing productivity and sustaining profit despite abrupt global market changes. The RMS is designed to have the best of both worlds – the high throughput of dedicated manufacturing lines (DML) and the flexibility of flexible manufacturing systems (FMS). It is intended to handle changes in production capacity and manufacturing requirements. Many approaches have been proposed in the academic literature to deal with the reconfiguration manufacturing paradigm. The well-known approaches are biological manufacturing systems (BMS) [6] and holonic manufacturing systems (HMS) [7].

A biological manufacturing system is a distributed manufacturing system, in which each part tries to achieve its own goals and each machine tries to attract them for processing. It has the capability of learning and healing itself when problems appear in various work stations and it is structured from the bottom up in a self-organised manner. BMS focuses on self-organising and evolution since it is the major strength of biological organism to keep cells alive.

A holonic manufacturing system is based on the concept of holon. A holon is an autonomous and co-operative building block of a manufacturing system for transforming, transporting, storing and/or validating information and physical objects. It consists of two parts: an information processing part and physical processing part. It can be part of another holon and has the ability to create and control the execution of its own plan and/or strategies. HMS is considered a system of holons that co-operate to achieve a goal or objective and integrates the entire range of manufacturing activities from order booking through design, production, and marketing.

Implementations of intelligent manufacturing systems rely on the use of techniques developed in the artificial intelligence, for example multi-agent techniques, formal knowledge representation and knowledge inference.

Semantic Web technologies have attracted lately great interest of industrial automation researchers. Lastra and Delamer [8] formulate one of the problems as follows: *How to enable two devices with no previous knowledge on each other's type, conceived using different paradigms and interaction models but still with complementary skill sets, to interact autonomously*?

IV. TEST CASE EXAMPLE

The following laboratory example illustrates some reconfiguration ideas and challenges of a modular machine which is a representative model of a simple work cell that can be found in various manufacturing environments. The machine is shown in Figure 1. It consists of four parts: Machine I (Processing Machine), Machine II (Handling Machine), Simple Conveyor Chain and Machine III (Filling Station). Each machine is built from some mechatronic devices for performing manufacturing or logistic operations.

The Machine I consists of four mechatronic devices: rotating indexing table, plunger, drill and kicker performing respectively rotating, hole checking, drilling and kicking. The Machine II consists of two devices: receptacle and picker. The receptacle performs presence detection and colour sensing while the picker performs picking and sorting, The simple conveyor chain consist of four independent conveyor loops and is used for moving the work pieces across the manufacturing floor. The Machine III consists of two tanks and a mixing valve. It is used for filling the supplied pieces with the liquid in its tanks.

This work cell supports multiple production scenarios [9] resulting from change of product, its features, availability of components, orders, etc. Figure **2** shows an example of supporting multiple sequences of production resulting from change in the availability of components. It illustrates the complying of two different scenarios with the same given requirement: hole testing followed by drilling then hole filling with a colour liquid (red or green) according to the colour of the work piece. In scenario (i), following the hole testing process, the rotating indexing table moves the work piece towards the drill. If the work piece does not have a hole, the drill drills a hole in the work piece only if the plunger has not drilled a hole then the rotating indexing table moves the work pieces towards the kicker in which it gets kicked to the receptacle. The picker picks the work piece after being detected and colour sensed by the receptacle and places the work piece on Conveyor IV which moves the work piece to Machine III, then Machine III fills the work piece with a coloured liquid. Whereas in scenario (ii), Conveyor IV is faulty which means the picker has to place the work piece on Conveyor I which moves the work piece to Conveyor II then from Conveyor II to Conveyor III and then Conveyor III moves the work piece to Machine III.

The addition of a new device or the replacement of a faulty device with a new one requires deactivation of the system. Moreover, after physically connecting and allocating the device, it may require the adjustments of some lower level controls, production plans and may require reprogramming parts of the control system. In order to recognise the operations and skills of a newly added device and identify any fault within any device in the work cell, a knowledge representation is needed a) to capture characteristics and relations among these devices b) to be capable of embedding theories of intelligent reasoning; c) to be capable of incorporating concepts from different manufacturing domains; d) to be capable of formally representing concepts and relations in which provides computability; and e) to be capable of describing the real world of manufacturing.

V. Ontological Knowledge Representation

The famous definition given by Thomas Gruber says that an ontology is a “formal specification of conceptualisation” [10]. An ontology specification is a formal description of a domain of interest, used for a particular purpose and expressed with a controlled vocabulary. The computer scientists have made a clear distinction between the terminological components (Tbox) and assertional components (Abox). The Tbox vocabulary specifies concepts that have associated Abox facts. Figure 3 shows the relationship between an Abox and Tbox. The combination of both represents a knowledge base for the domain [11]. In our case the manufacturing industry is our domain of interest; Tbox describes the concepts in the manufacturing industry whereas Abox represents the instances of these concepts.

The knowledge of the manufacturing environment can be formalised along with the rules and the relationships between the objects. The reason behind formalising is to form a reliable and interoperable description of the manufacturing s domain. Hence ontologies can be used as a means of formalising this knowledge using a common vocabulary or a standard common language like OWL to ensure interoperability.

There were a few efforts towards forming a common manufacturing ontology during the last decade such as the work [13] done by the National Institute of Standards and Technology, followed by a recent proposal of a common manufacturing upper ontology for manufacturing system (MASON). The proposal was built upon three head concepts: entities, operations and resource [14]. The principle behind that is to provide shared and common understanding of the manufacturing domain. MASON has already been used across many useful applications such as multi agent systems for manufacturing [15].

VI. Knowledge Model

The knowledge model of our simple manufacturing environment is expressed using OWL – Web Ontology Language developed by W3C (World Wide Web Consortium). OWL is a language used for defining and instantiating web ontologies. OWL’s interoperability is achieved by the use of the standardized eXtensible Markup Language (XML). OWL supports knowledge sharing and reuse which is very important for adding new facts to the knowledge model and keeping it up to date [16],[17]. These features have been recently utilised in forming an OWL representation of the Automation Object Reference model for Industrial-Process Measurement and Control Systems [18],[19],[20].

OWL-DL, one of the OWL dialects, is used for expressing our knowledge model for the reason that it ensures reasoning will be computable and decidable. Protégé [21], an open source platform for creating knowledge models, is used for creating the Tbox (formal specification of the environment) and the Abox (instances of machines that comply with Tbox) of our environment. Each machine has an OWL document with instances of all the class (operations and resources) propertied related to the machines. Our knowledge model is based on MASON. Figure 4 shows the main important head concepts that have been identified: Resource, Operation with the addition of Controller.

The Controller concept is important in terms of identifying the distributed controllers of the system. The experimental testbed described above is controlled by distributed control devices compliant with the new IEC 61499 standard [22], [23]. Correspondingly, the Controller component of the knowledge model includes descriptors, defining a specific run-time platform support of IEC 61499 function blocks. In our case-study only two run-time platforms have been specified: FBRT [24] and some other (non-FBRT). The Operation concept consists of two sub-concepts: Manufacturing Operation and Logistic Operation. Most of the time, the logistic operations are acting as a link between manufacturing operations.

Table 1 shows some of the important properties that we identified in the environment based on MASON. These properties play a crucial role in helping in inferring useful facts about the environment.

TABLE I Important properties in the sample manufacturing environment.

Property	Description
‘nextOperation’	This property determines the next feasible operation that can follow the current operation.
‘requiresTool’	This property links Operation with its Tool(s). Its opposite property is ‘requiredToolFor’
‘isLastOperation’	This property classifies the feasible last operation(s) in a machine
‘isStartUpOperation’	This property classifies the feasible start-up operation(s) in a machine
‘controlledBy’	This property links the tool with the embedded level controller. Its opposite property is ‘controles’
‘requiresMachine’	This property links Machine with its Operation(s). Its opposite property is ‘enableRealisationOf’

Figure 5 shows some instances (Abox facts) of our manufacturing system and the relationship between them. The two main important instances are the processing machine instance (Processing_machine_1) and handling machine instance (Handling_machine_1). The other instances are the operations instances performed by these machines. As seen in this figure, the ‘nextOperation’ property links the operations forming a graph like structure. The decision engine of the configuration agent uses this structure in the process of generating a new configuration. There are other instances which are not shown in this figure like instances of the tools that perform these operations.

In order to support the additions of new devices to the environment, the knowledge model needs to facilitate growth in terms of concepts and relationships. For that reason, all the main concepts and relationships of the manufacturing domain is included in the upper ontology OWL document of the knowledge model. At the same time, each machine has an OWL ontology document that extends the upper manufacturing ontology to include more specific sub-concepts and relations related to the machine and instances of the objects related to the machine such as the machine, tools, operations, etc. Therefore, the upper ontology document consists of Tbox only while the machines OWL documents consist of both parts: Tbox and Abox.

VII. Reconfiguration Agent

The proposed ontology-based reconfiguration agent is an intelligent software agent that allows the manufacturing system to adapt changes in the manufacturing requirement and/or environment. It generates an alternative or a new feasible configuration that satisfies the requirements. The new configuration consists of a sequence of manufacturing operations which are available in the manufacturing floor. The software agent decides the new manufacturing configuration using the inferred facts about the manufacturing environment ontological knowledge model, the extracted information about the floor specification, and the given manufacturing requirements.

The reconfiguration agent performs three main tasks: (i) it checks whether a requested operation is supported by the manufacturing environment, or not; (ii) it generates a new configuration by forming a sequence of logistic operations that inter-connects the requested operations together; and (iii) it divides the new generated configuration into sub-configurations according to the distribution of the controllers; and (iv) it provides feedback of the reason behind any task failure if any of the three tasks are not completed.

A. Architecture

The reconfiguration agent has a multi-layered architecture consisting of three layers: the intelligent decision layer, the analysing and modelling layer, and the specification layer. Each layer depends on the layer beneath it and uses its capabilities and features. At this same time, each layer consists of components that are responsible of carrying out specific activities and/or providing specific functions.

Figure 6 shows all the three layers of the reconfiguration agent and their components. The architectural division of the layers is based on the specialisation level of the layer. The first layer explicitly provides all the knowledge and specification needed for decision making. The second layer extracts the given explicit information, transforms it into an implicit format, and then analyse it and model it for easy access and manipulation. It extracts and creates a list of the requested operation from the requirements specification; a list of the available machines in the manufacturing floor from the floor specification; and an ontological knowledge model from the environment knowledge specification. The third layer is where the intelligent reasoning takes place which includes generating final configuration, dividing the generated configuration into sub-configurations according to the distribution of the controllers.

The most essential components of the agent are: Knowledge Modeller, Decision Engine, and Deployment Manager.

B. Implementation

The prototype of our reconfiguration agent has been implemented using Java platform in conjunction with Eclipse IDE. The main reasons behind using Java platform are: (i) the accessibility of frameworks that support programmatic environment for OWL; and (ii) the availability of OWL reasoners for knowledge inferring and decisive reasoning. These frameworks and reasoners can ease and speed up the prototyping process of the reconfiguration agent.

Jena [25]- an open source Java framework that provides a programmatic environment for OWL - has been used for creating the ontological knowledge model of the manufacturing environment. It constructs a tree-like graph of the knowledge from reading the given machine's OWL documents and provides methods that enable access, update and manipulate the knowledge model.

Pellet [26], an open-source Java based OWL-DL reasoner is used as our rule-based reasoner to work in conjunction with Jena framework. It has been used by the decision engine and deployment manager for inferring facts about our ontology model of the environment. Also, it has been used by the knowledge modeller for the merging process of the machines' knowledge and for checking the consistency of the OWL documents.

JDOM [27], a Java representation of XML documents is used by the requirements analyser and floor analyser components for reading the manufacturing requirements XML document and floor specification XML document.

C. Requirements Analyser

Requirements analyser extracts all the required operations from the Requirements XML document. An example of the requirements is as follows:

```
<Requirements>
 <ReqOperations>
   <ReqOperation>
      Hole_testing
   </ReqOperation>
   <ReqOperation option = {depth: 2cm, speed = 12}>
      Drilling
   </ReqOperation>
   <ReqOperation option = {criteria: hole_presence}>
      Sorting
   </ReqOperation>
 </ReqOperations>
</Requirements>
```

The root element of the requirements document is **Requirements**. For the sake of simplicity, we only considered Requested Operations as the only element of requirements. It consists of at least one or more Requested Operation element. The options attribute of the Requested Operation is used to state further specification of the requested operation. Different requested operation has different options. E.g. Drill options can include the depth of the hole and speed of the drill while Sorting options can include the criteria of the sorting methodology. The requirement analyser creates an ordered list of required operations object. Each object contains the name and the options of the requested operation. This list is used later on by the decision engine component of the system.

D. Floor Analyser

The floor analyser captures all the needed floor specification about the machines in the manufacturing system from the floor specification document. This includes the location, orientation, OWL ontology document of each machine and the connections of one machine to another.

An example is as follows:

```
<FloorSpecification>
 <Machines>
  <Machine ID=1 initial=true pos={x0:2 y0:2 x1:2 y1:2}>
    <MachineOntology>
      http://localhost/ProcessingMachine.owl
   </MachineOntology>
   <ConnectedTo startupOp = "Rotation" >2</ConnectedTo>
  </Machine>
  <Machine ID = 2 initial=false pos={x0:2 y0:2 x1:2 y1:2}>
    <MachineOntology>
      http://localhost/HandlingMachine.owl
    </MachineOntology>
    <ConnectedTo startupOp = "Picking,
                Presence_detection" >
      1
    </ConnectedTo>
  </Machine>
 </Machines>
</FloorSpecification>
```

Floor Specification element is the root element of the document. It consists of the **Machines** element and the **Ontologies** Element. The **Machines** element can contain one or more Machine element depending on the number of machines in the manufacturing floor. The position attribute of the machine elements is used to determine the location and the orientation of the machine. For the sake of simplicity, we assumed that any machine in our environment can fit in a squared shape frame. ‘x0’ and ‘y0’ determines the position of the top left hand corner point of the machine while the ‘x1, and y1’ is the position of the bottom right corner point of the machine. The ID attribute of the machine is used for identifying the machine while the initial attribute is used to determine the startup machines. The Machine element contains two sub- elements: one MachineOntology and at least one ConnectedTo Elements. The MachineOntology has the URL of the machine’s OWL document while the ConnectedTo states the machines that this machine connects to. The attributes of ConnectedTo determine the startupOperation in that connection. The floor analyser creates a graph of the available machines objects and provides methods for accessing this graph. Each object includes all the extracted information (location and Ontology URL). This list is used later on by the knowledge modeller component.

C. Knowledge Modeller

The knowledge modeller constructs ontology knowledge model of the manufacturing environment using the machines list extracted by the floor analyser. This list contains vital information about the machines for the operation of the knowledge modeller: the URL of the OWL document and the location of each machine. The modeller uses Jena model factory and Pellet inference engine (given by the decision engine) to construct an Ontology model of the factory floor by reading each machine’s OWL document. The model factory converts each OWL document to a graph which is the primary data structure in Jena. This graph is wrapped by a model that provides convenient methods for the decision engine to access the contents of the graph.

D. Decision Engine

The decision engine produces a feasible configuration that satisfies the given requirements using the floor graph and the ontology knowledge model. First it uses Pellet and Jena to connect the last operations with the initial operations of the machines in the factory floor via the ‘nextOperation’ property then starts to inspect whether the requested operations exist in our ontology knowledge model. If all of them exist, it tries to find the logistic operations that can connect the requested operations with each other. It accesses the first satisfied requested operation and marks it as current operation. At the same time it performs a search until it finds the next satisfied requested operation. If it can not find one or a series of logistic operations that connects the current operation to the next one, it indicates that it can not connect the two operations. After that it moves to the next satisfied operation and tries to do the same searching process until it finds next requested operation.

Figure 7 shows an example of two different requirements and the results produced by the decision engine. The facts about our environment (see Figure 5) show that the drilling operation comes after the hole testing and not vice versa. We were able to infer that by performing a search through the graph-like structure of the operations. The ‘Requirements I’ can be satisfied because the operations are available and a logistic path exists between them while the ‘Requirements II’ can not be satisfied because there is no logistic connection between hole testing and drilling. Also this figure shows that the system can produce two different configurations in the (Drilling_Sorting section) and (Hole_testing - Sorting section), since the decision engine concluded that the picker can pick the workpiece from two different locations (the receptacle and rotary_hole of the rotating table) as both are within the given range of picking in the ontology.

E. Deployment Manager

After successfully generating a new configuration of the system, the deployment manager deploys the new configuration on the distributed controllers of the system. The deployment process of the new configuration is not a straight forward process for the reason that different devices are controlled by different controllers. Thus the deployment manager uses the knowledge model of the manufacturing environment to determine the tools that execute the operations in the new configuration and then it identifies the controllers that control these tools. After that it divides the new configuration into sub-configurations according to the distribution of the controllers and then deploys the sub-configuration on the right controller.

Figure 8 shows an example of division of the new configuration (Hole_testing_1 → Rotation_2 → Drilling_1 → Rotation_3 → Kicking_1 → Picking_1 → Sorting_1) into two sub configuration: sub-configuration I (Hole_testing_1 → Rotation_2 → Drilling_1 → Rotation_3 → Kicking_1) and sub-configuration II (Picking_1 → Sorting_1). In the example, the deployment manager determines all the tools that execute the operations of the configuration via ‘requiresOperation’ property and then identifies the controllers of the tools via the ‘controlledBy’ property. Subsequently, it divides the generated configuration into sub-configurations according to the identified controllers (Machine_I_Cntrl, Machine_II_Cntrl) and then distributes and deploys the new sub-configurations across the low level controllers.

The proposed low level controllers of our system have to be in compliance with the IEC 61499 function block standard [22], which provides modular design with well-defined interfaces, portability and transparent mapping of applications to different hardware configurations.

The detailed description of the low-level control architecture is beyond the scope of this paper due to the space constraints. In short, in our proposed design, each controller consists of a function block for each device, player function block, and object transfer adapter function block. Each device function block encapsulates all the functionalities of the device. The player function block is connected with deployment manager and also with all the function blocks of the devices, in order to deploy the given sub-configuration in the controller. The object transfer adapter controls the transfer protocol between the controllers. Figure 9 shows the proposed low level controllers of our system.

VIII. Future Work

The future work can be categorised into three different categories. The first category involves improving the current reconfiguration agent prototype in terms of decision making skills, architecture, analysing, etc. The second category involves improving the use of ontology in representing the manufacturing environment and manufacturing specification. The third category includes integrating the reconfiguration agent with the embedded controllers.

Improving the current prototype involves improving the problem solving skills of the agents in terms of solving more complex problems. One way of improving this is extending the architecture of the decision engine or adding new components that can fully utilise OWL-DL knowledge inferring and decidable reasoning capabilities.

The ontological knowledge model of our simple manufacturing environment can be improved with the addition of floor specifications to the current knowledge model instead of having it in a separate XML file.

At the same time requirement specifications can be represented using ontology. This will enable the agent to reason and infer knowledge about the specifications.

Our current research is moving towards integrating the reconfiguration agent with the distributed controllers of our system's low level components. This will enable the agent to interact and communicate directly with the controllers of the components for the purpose of immediately deploying the new configuration. The distributed low level controllers of our system have been developed in compliance with the IEC 61499 standard.

IX. CONCLUSION

The developed ontology-based reconfiguration agent is an intelligent reasoning software agent that allows the manufacturing system to adapt to changes in the manufacturing requirement and/or environment by generating an alternative or a new feasible configuration. The new configuration consists of a sequence of manufacturing operations that is available in our manufacturing floor and satisfies the current manufacturing requirement. The software agent intelligently decides the new manufacturing configuration using the inferred facts about the ontological knowledge model of the environment and the extracted information about the floor specification and the given manufacturing requirements.

The agent prototype has been implemented using Java platform in conjunction with Eclipse IDE. Jena - an open source java framework that provides a programmatic environment for OWL - has been used for representing the ontological knowledge model of the manufacturing environment. Pellet - an open-source Java based OWL-DL reasoner - has been used as a rule-based reasoner to work in conjunction with Jena framework.

The prototype of the reconfiguration agent has been used to solve small reconfiguration problems. The agent uses the knowledge model to infer knowledge about the environment. Subsequently and according to the derived facts from the knowledge model, it generates solutions to these reconfiguration problems if possibly achieved. Each solution consists of new feasible configuration which is divided by the agent into sub-configurations according to the distribution of the embedded controllers in the system.

ACKNOWLEDGMENT

This work has been supported in part by the research grants 3607207/9273 and 3607028/9273 of the University of Auckland.

REFERENCES

[1] Leitão, P., “An Agile and Adaptive Holonic Architecture for Manufacturing Control”, PhD Thesis, University of Porto, 2004.

[2] V. Vyatkin, "Intelligent mechatronic components: control system engineering using an open distributed architecture," *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA '03. IEEE Conference*, vol. 2, pp. 277-284 vol.2, 2003

[3] Alsafi, Y., Vyatkin V., An Ontology-based Reconfiguration Agent for Intelligent Mechatronic Systems, Lecture Notes in Computer Science, Vol. 4659, pp. 114-126

[4] Koren, Y., Heisel, U., Jovane, F., Moriwaki, T., Pritschow, G., Ulsoy, G., Van, Brussel, H. (1999) Reconfigurable manufacturing systems. CIRP Ann 48(2), pp. 527–540

[5] Koren, Y., General RMS Characteristic. Comparison with Dedicated and Flexible Systems, in Reconfigurable Manufacturing Systems and Transformable Factories, edited by Dashchenko, Netherlands, Springer-Verlag, pp.1-13

[6] Ueda, K., Vaario, J., Ohkura, K. (1997). Modelling of biological manufacturing systems for dynamic reconfiguration. Ann. CIRP 46, pp. 343–346.

[7] Markus, A., Vancza, K., Monosstori, L. (1996). A market approach to holonic manufacturing. Ann. CIRP 45, 433–436.

[8] Martinez Lastra, J. L., Delamer, I. M., Semantic Web Services in Factory Automation: Fundamental Insights and Research Roadmap, IEEE Transactions on Industrial Informatics, Vol. 2, No. 1, February 2006

[9] Sardesai, A.R., Mazharullah, O., Vyatkin, V., Reconfiguration of Mechatronic Systems Enabled by IEC 61499 Function Blocks, Proceedings of the 2006 Australasian Conference on Robotics & Automation

[10] Gruber, T. R. A Translation Approach to Portable Ontology Specifications, Knowledge Acquisition,5 : 199-220, 1993.

[11] Lacy, L., OWL: Representing Information Using the Web Ontology Language, Trafford Publishing, 2005

[12] Saad, S.M. (2003) The reconfiguration issues in manufacturing systems. Journal of Materials Processing Technology, 138, pp. 227-283

[13] Schlenoff, C., Ivester, R., Knutilla, A.,. A robust process ontology for manufacturing systems integration. In Proceedings of 2nd International Conference on Engineering Design and Automation, 1998.

[14] Lemaignan, S., Siadat, A., Dantan, J.-Y., Semenenko, A.. MASON: A Proposal For An Ontology Of Manufacturing Domain Distributed Intelligent Systems. Proceedings of International IEEE Workshop on Distributed Intelligent Systems (DIS 2006), Collective Intelligence and Its Applications. pp.195 – 200.

[15] Obitko, M., Marik, V., Ontologies for multi-agent systems in manufacturing domain, In DEXA'02: Proceedings of the 13th International Workshop on Database and Expert Systems Applications, pages 597–602, Washington, DC, USA, 2002.IEEE Computer Society.

[16] Web Ontology Language (OWL), W3C Recommendation http://www.w3.org/TR/owl-ref/

[17] Aref, M. M., Zhou, Z., The Ontology Web Language (OWL) for a Multi-Agent Understating System, 2005

[18] Automation Objects for industrial-process measurement and control systems - IEC SB3/TC 65, Working draft, 2002

[19] Automation Objects: Special issue of Int. J. Manufacturing Research, Brennan, R.W., Ferrarini, L., Lastra, J.M. and Vyatkin, V. (Eds), Vol. 1, No. 4, pp.379–381.

[20] Martinez Lastra J.L., López Orozco O. J., Semantic Extension for Automation Objects, IEEE Conference on Industrial Informatics, Proceedings, Singapore, 2006

[21] Stanford Protégé Website, 2004. The Protégé Project. (http://protege.stanford.edu/)

[22] IEC 61499 - Function blocks for industrial-process measurement and control systems - Part 1: Architecture, International Electrotechnical Commission, Geneva, 2005

[23] Vyatkin, V., IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design, 300 pp., O3NEIDA - Instrumentation Society of America, 2007

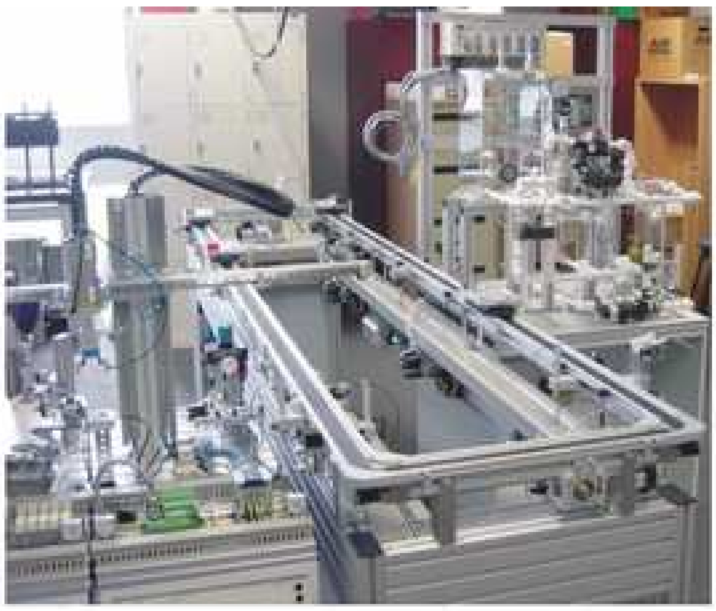
[24] Function Block Run-time (FBRT), a part of Function Block Development Kit (FBDK), Online, www.holobloc.com

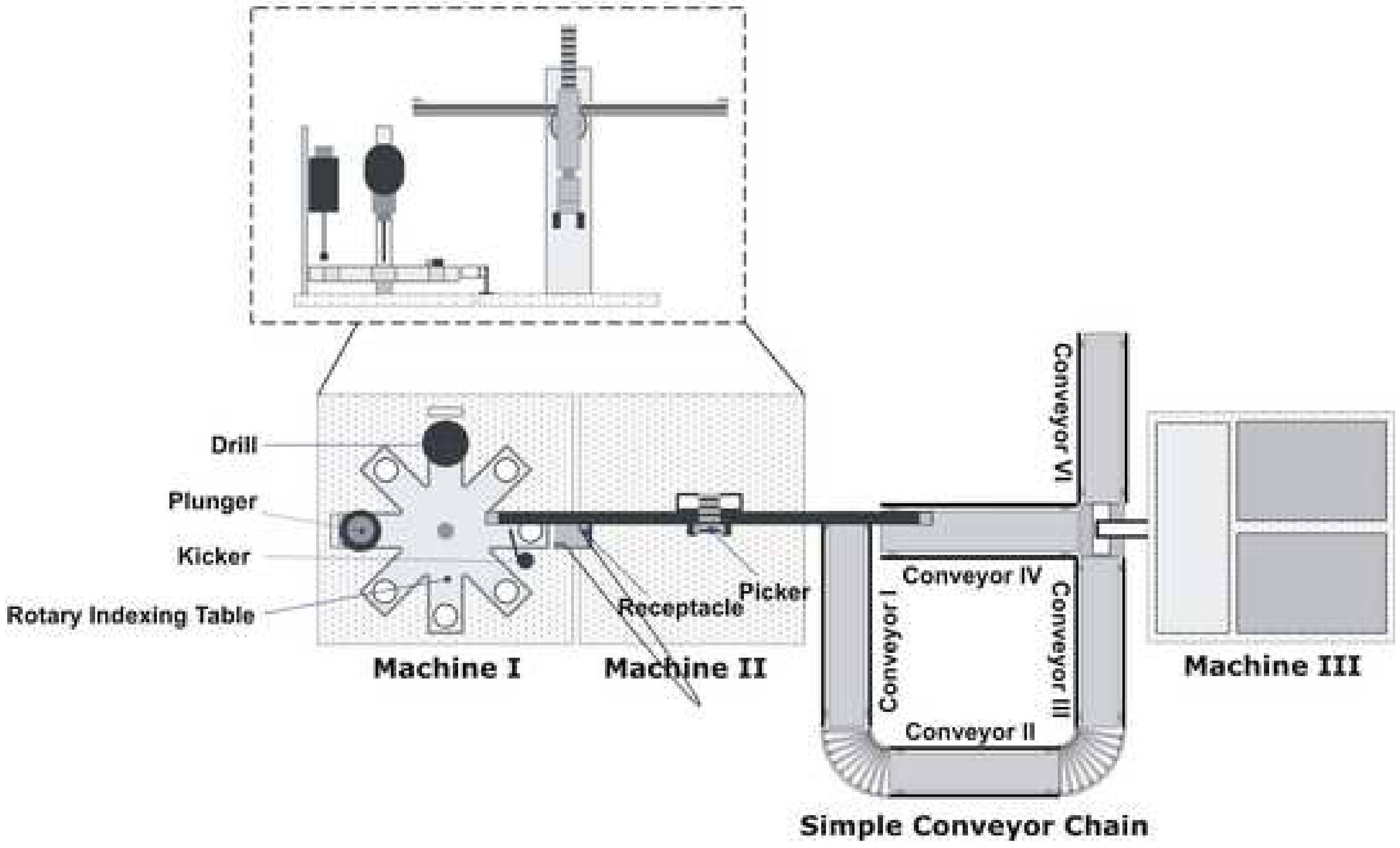
[25] Jena A Semantic Web Framework for Java http://jena.sourceforge.net

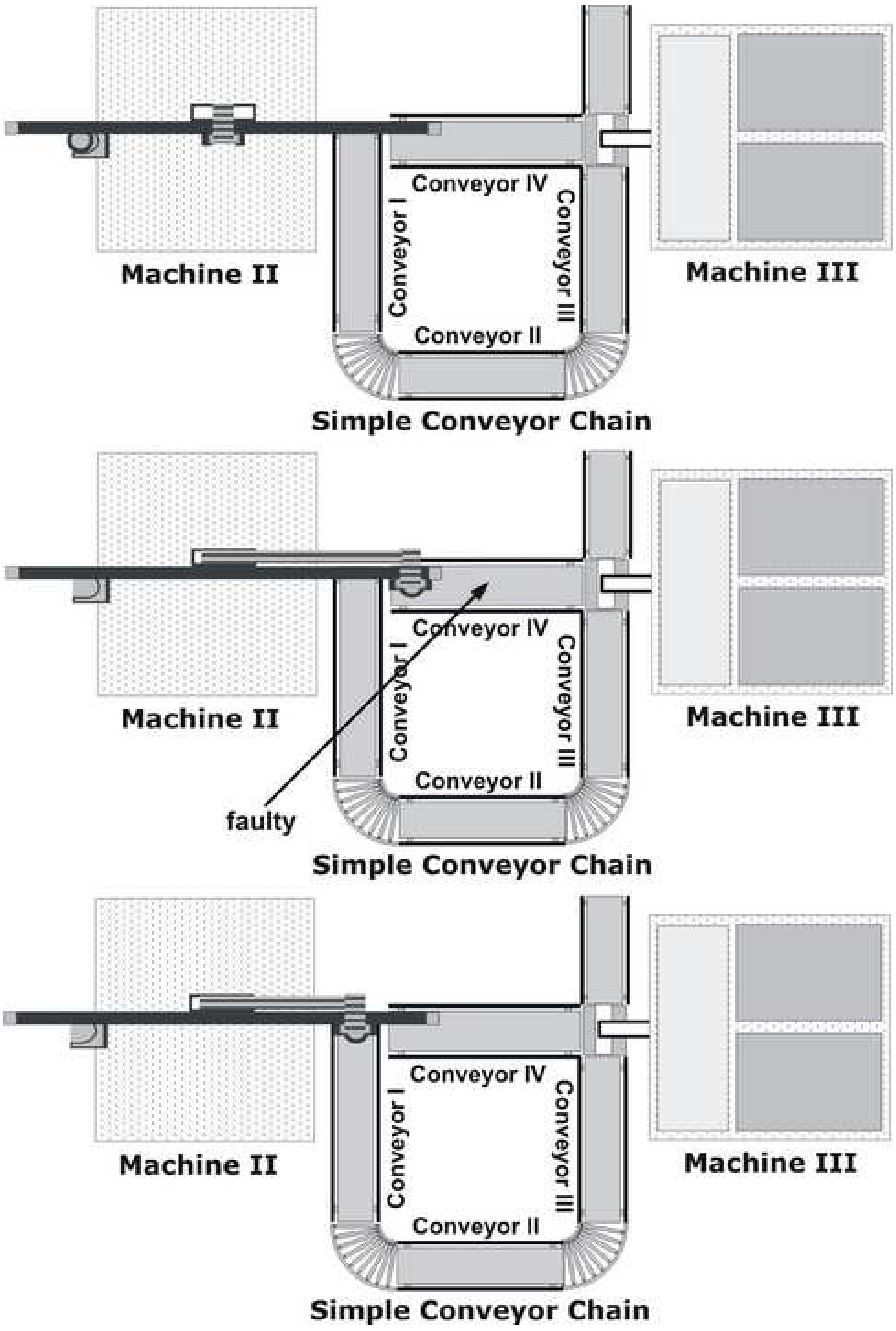
[26] Pellet: An OWL-DL Reasoner (http://pellet.owldl.com/)

[27] JDOM - (Java Document Object Model) Java Toolkit for working with XMLFB Book (http://www.jdom.org)

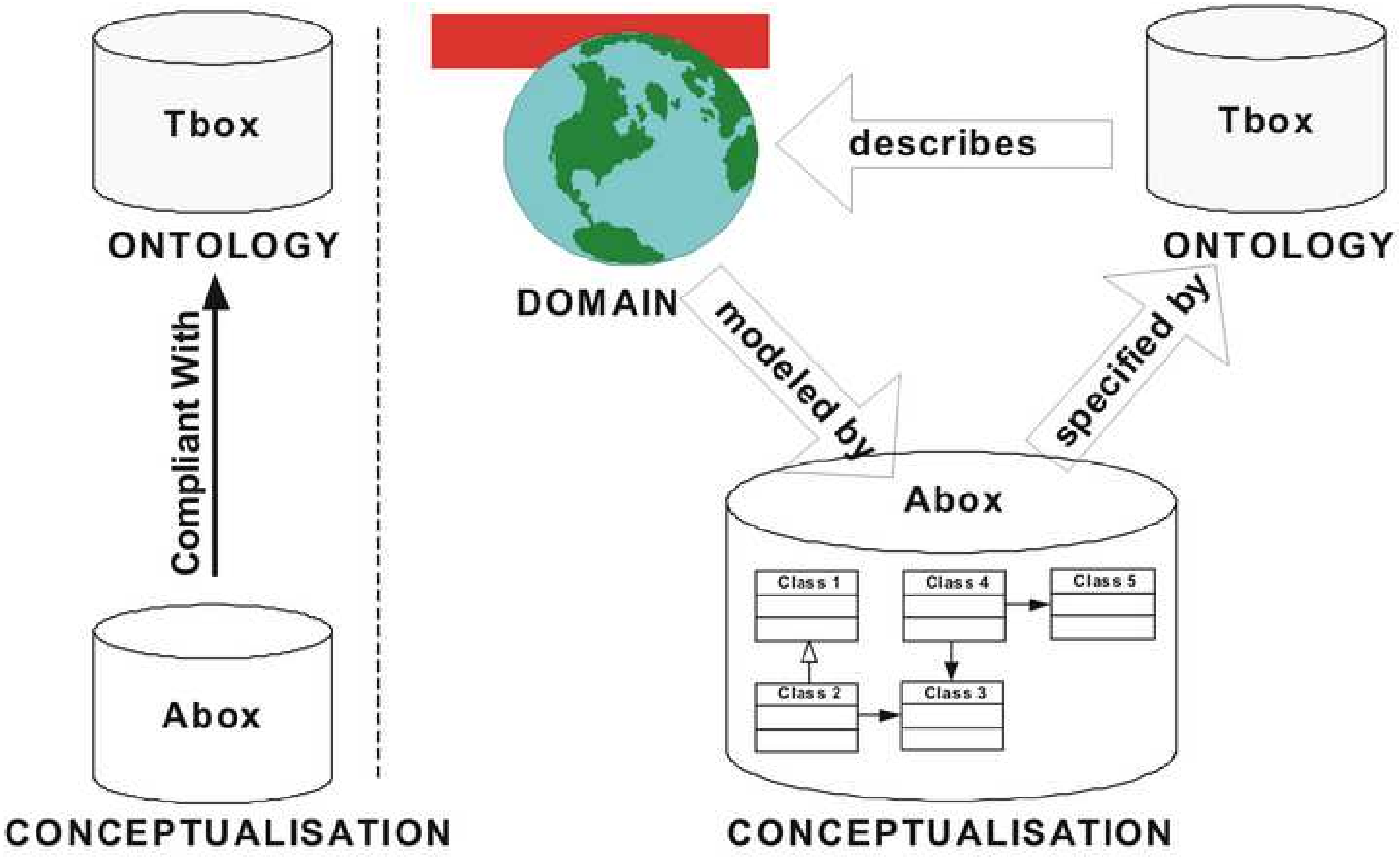
List of figures and their captions

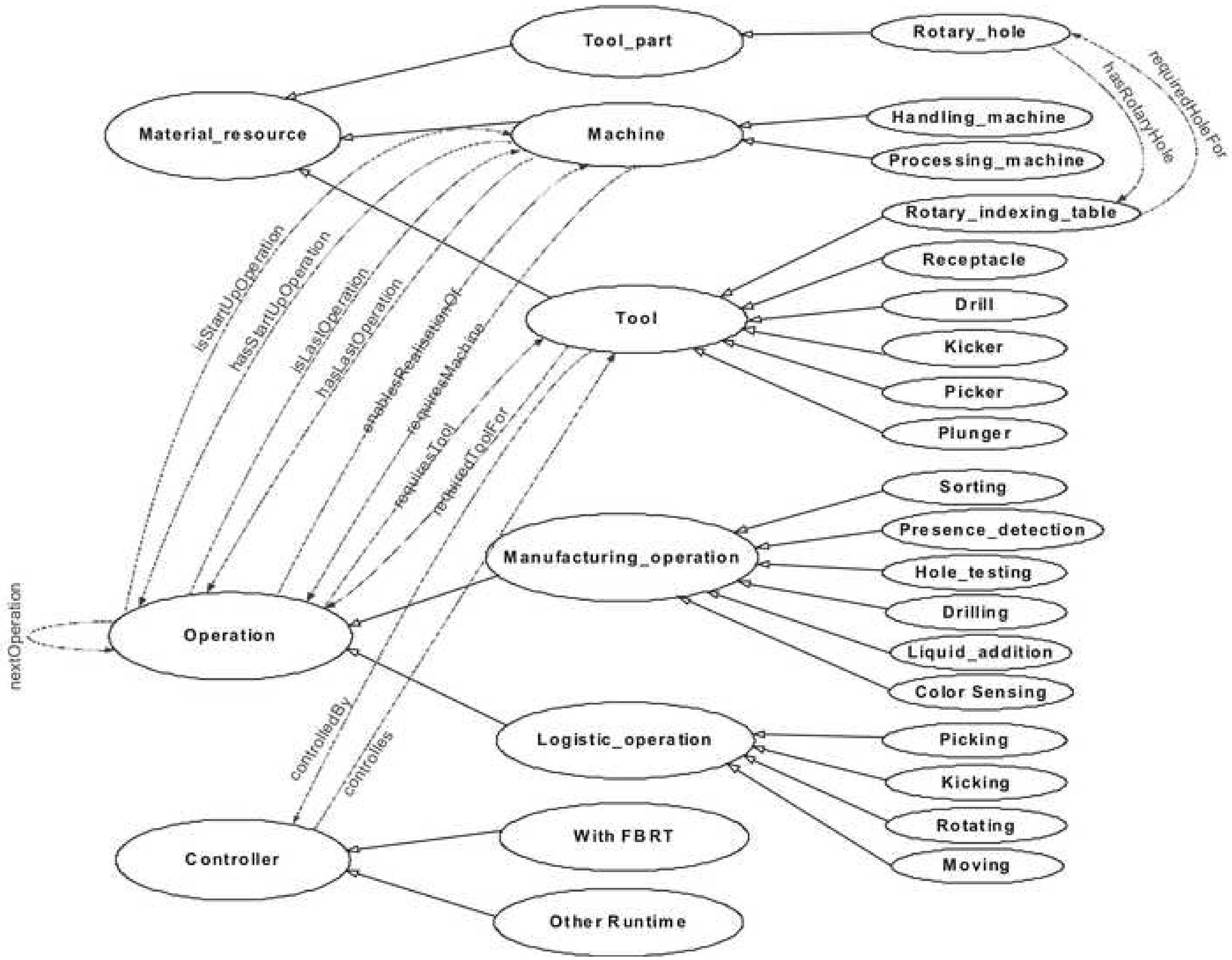



Drill
Plunger
Kicker
Rotary Indexing Table
Receptacle
Picker
Machine I
Machine II
Conveyor VI
Conveyor IV
Conveyor I
Conveyor III
Conveyor II
Machine III
Simple Conveyor Chain

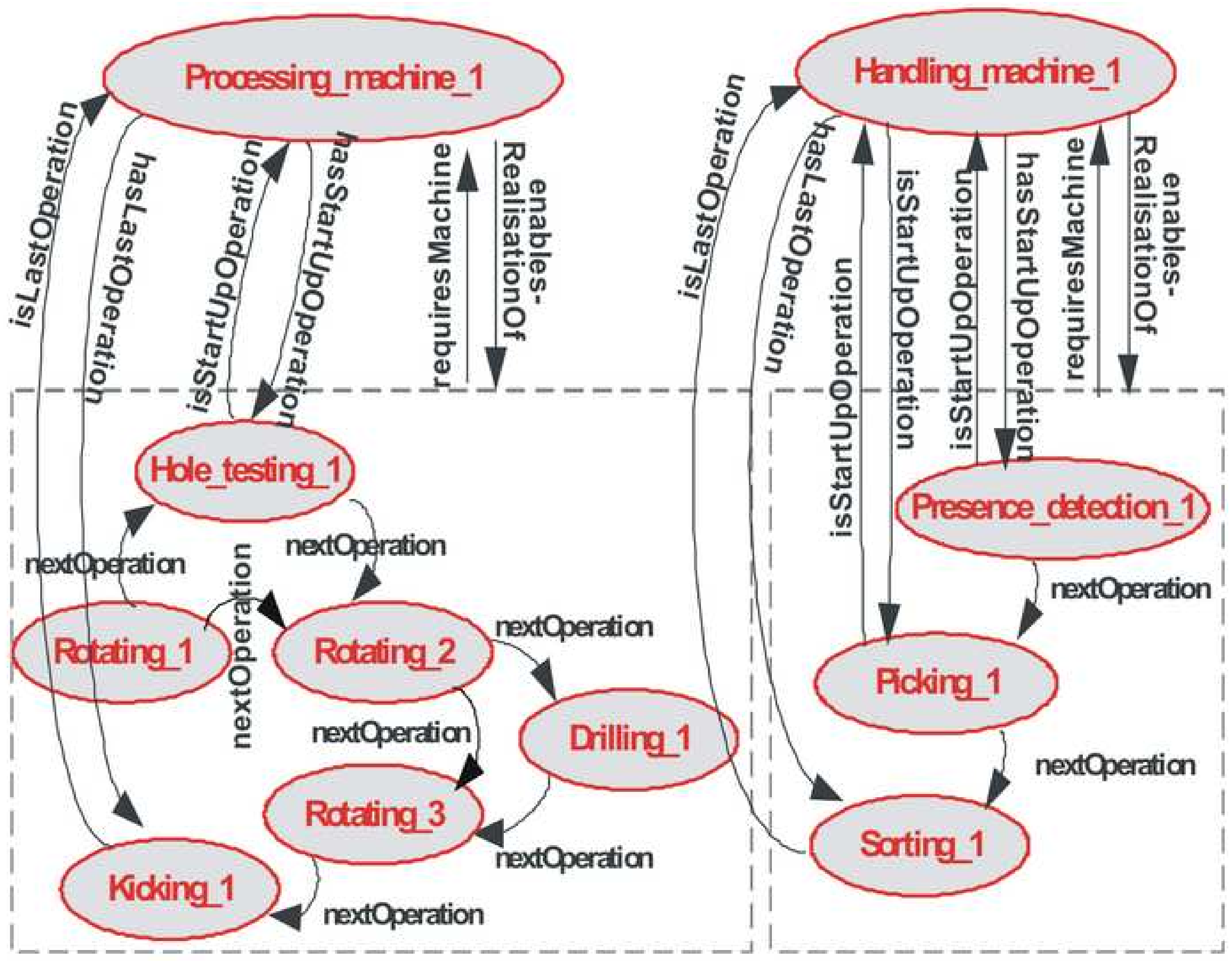

Machine II
Machine III
Conveyor IV
Conveyor I
Conveyor III
Conveyor II
Simple Conveyor Chain
Machine II
Machine III
Conveyor IV
Conveyor I
Conveyor III
Conveyor II
faulty
Simple Conveyor Chain
Machine II
Machine III
Conveyor IV
Conveyor I
Conveyor III
Conveyor II
Simple Conveyor Chain

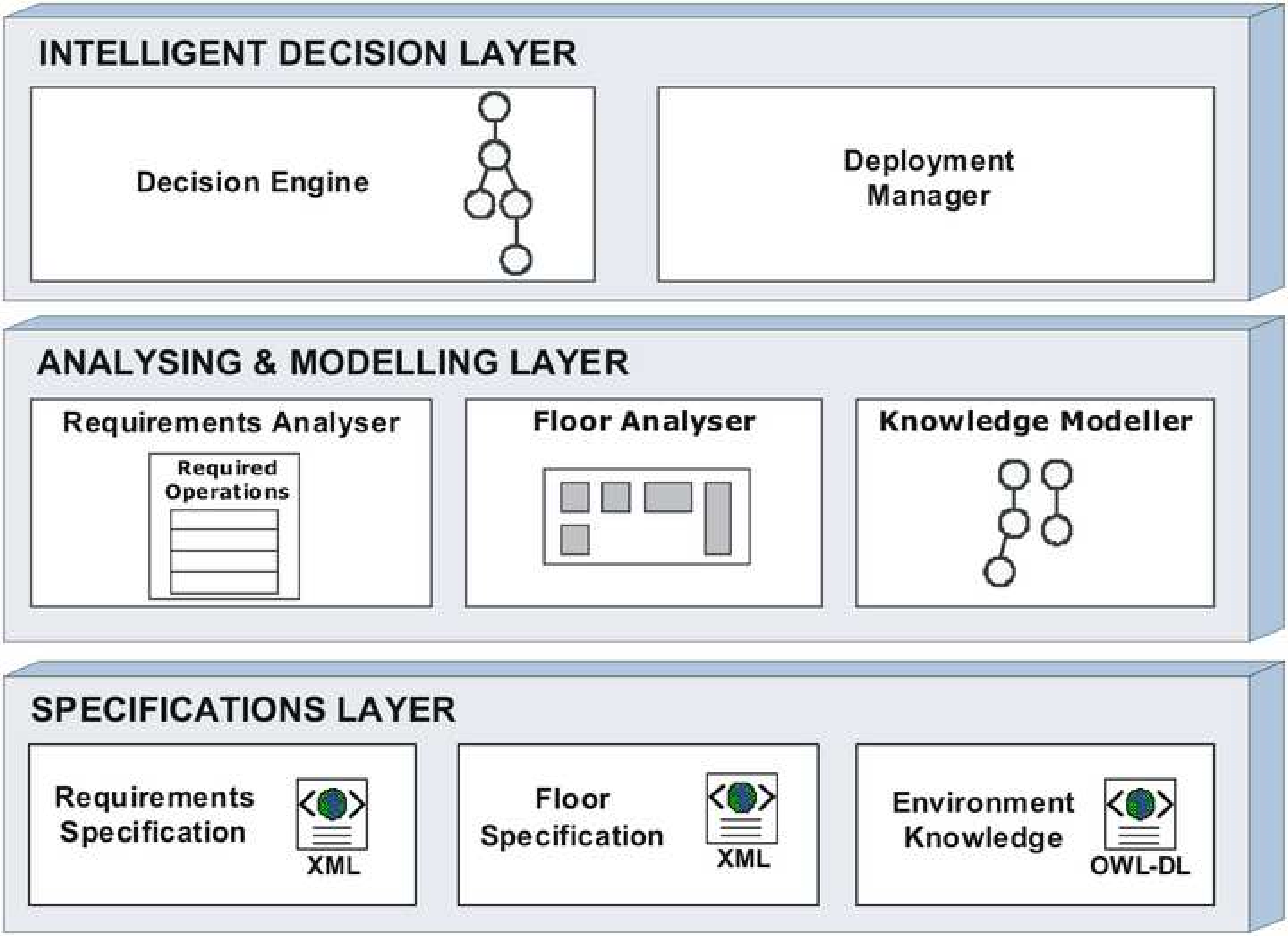
Click here to download high resolution image

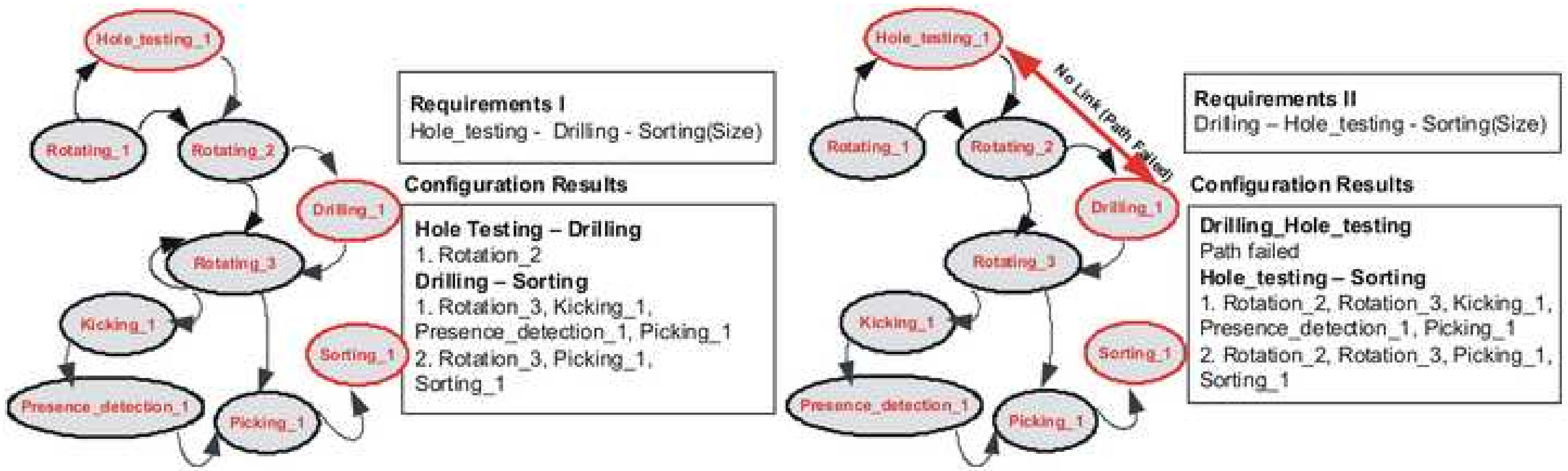



Tool_part
Rotary_hole
Material_resource
Machine
Handling_machine
Processing_machine
Rotary_indexing_table
Receptacle
Drill
Kicker
Picker
Plunger
Tool
Sorting
Presence_detection
Hole_testing
Drilling
Liquid_addition
Color Sensing
Manufacturing_operation
Operation
Logistic_operation
Picking
Kicking
Rotating
Moving
Controller
With FBRT
Other Runtime
hasRotaryHole
requiredHoleFor
isStartUpOperation
hasStartUpOperation
isLastOperation
hasLastOperation
enablesRealisationOf
requiresMachine
requiresTool
requiredToolFor
nextOperation
controlledBy
controlles

Click here to download high resolution image




INTELLIGENT DECISION LAYER
Decision Engine
Deployment Manager
ANALYSING & MODELLING LAYER
Requirements Analyser
Required Operations
Floor Analyser
Knowledge Modeller
SPECIFICATIONS LAYER
Requirements Specification
XML
Floor Specification
XML
Environment Knowledge
OWL-DL



Requirements I
Hole_testing - Drilling - Sorting(Size)

Configuration Results

Hole Testing – Drilling
1. Rotation_2

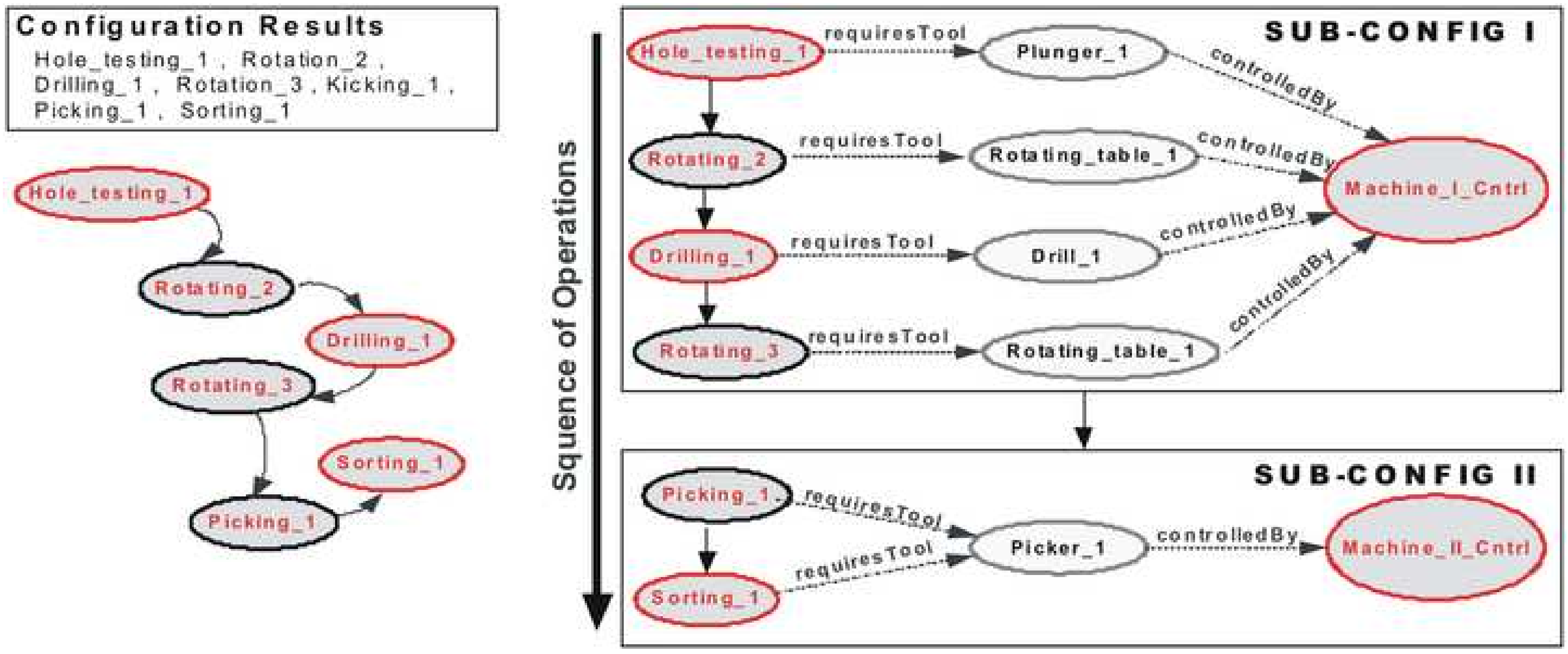
Drilling – Sorting
1. Rotation_3, Kicking_1, Presence_detection_1, Picking_1
2. Rotation_3, Picking_1, Sorting_1

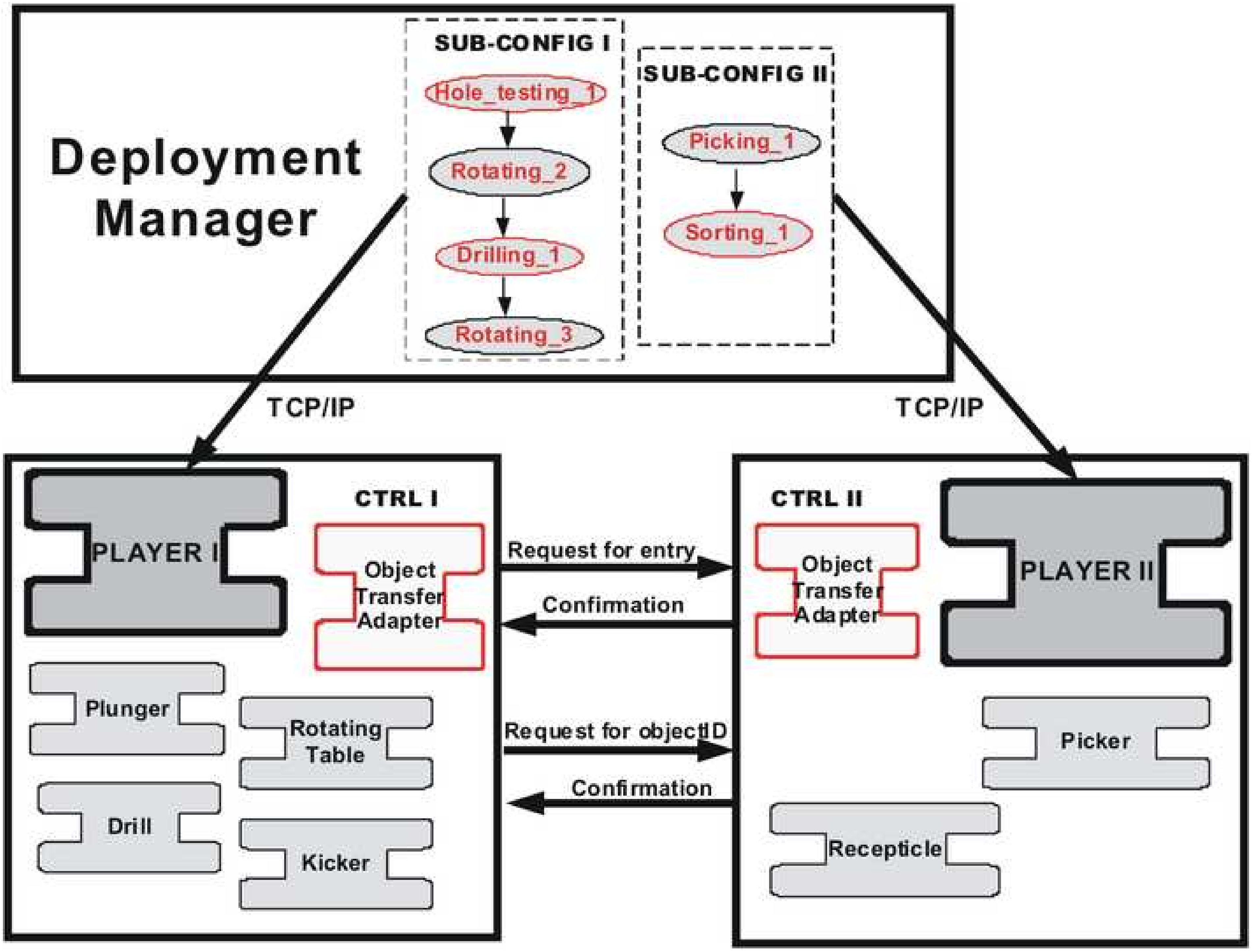
Requirements II
Driling – Hole_testing - Sorting(Size)

Configuration Results

Drilling_Hole_testing
Path failed

Hole_testing – Sorting
1. Rotation_2, Rotation_3, Kicking_1, Presence_detection_1, Picking_1
2. Rotation_2, Rotation_3, Picking_1, Sorting_1


Configuration Results
Hole_testing_1 , Rotation_2 , Drilling_1 , Rotation_3 , Kicking_1 , Picking_1 , Sorting_1
Hole_testing_1
Rotating_2
Drilling_1
Rotating_3
Sorting_1
Picking_1
Squence of Operations
SUB-CONFIG I
Hole_testing_1
requiresTool
Plunger_1
controlledBy
Rotating_2
requiresTool
Rotating_table_1
controlledBy
Machine_I_Cntrl
Drilling_1
requiresTool
Drill_1
controlledBy
Rotating_3
requiresTool
Rotating_table_1
controlledBy
SUB-CONFIG II
Picking_1
requiresTool
Picker_1
controlledBy
Machine_II_Cntrl
Sorting_1
requiresTool


Deployment Manager
SUB-CONFIG I
Hole_testing_1
Rotating_2
Drilling_1
Rotating_3
SUB-CONFIG II
Picking_1
Sorting_1
TCP/IP
TCP/IP
CTRL I
PLAYER I
Object Transfer Adapter
Plunger
Rotating Table
Drill
Kicker
Request for entry
Confirmation
Request for objectID
Confirmation
CTRL II
Object Transfer Adapter
PLAYER II
Picker
Recepticle